Fast Optimal Target Allocation, Internetted Fires, and the Value of Information

By David Olwell
Operations Research Department, Naval Postgraduate School, Monterey CA 93943
831-656-3583 (tel)
831-656-2595(fax)
dholwell@nps.navy.mil

and Alan Washburn
Operations Research Department, Naval Postgraduate School, Monterey CA 93943
831-656-3127 (tel)
831-656-2595(fax)
washburn@nps.navy.mil

DAS XII
April, 2004

ABSTRACT

The design of the Army's future combat system is predicated on the assumption that one can safely trade increased situational awareness for firepower and survivability. Increased global awareness is asserted to allow the Army to construct lighter deployable units while still maintaining equivalent or better combat capability. However, as we show, even perfect situational awareness and internetted fires do not *always* contribute to significantly improved combat lethality. We propose a partial framework for assessing the contributions of situational awareness, and show how this framework can be extended to aid combat planning. We have found instances where global use of information can achieve the same results as local use of information, but at a cost of only half the number of weapons. We have also found instances where there is no significant advantage to global use of information. We explain some of the factors that influence the difference.

EXECUTIVE SUMMARY

To assess the value of information and internetted fires, we must first model fire allocation schemes, and how they depend on situational awareness. It is sometimes suggested that warfare can be conducted by autonomous "agents" that operate and make decisions based only on locally available information. It is also sometimes suggested that modern armed forces should attempt to achieve a C4ISR system sufficiently sophisticated to achieve "internetting-of-fires", since it is only with such a system that the capabilities of long-range weapons can be exploited to the fullest. These suggestions are in conflict, since the one asserts that local optimization is sufficient or at least inevitable, while the other asserts that only global optimization will do. In an abstract way, these ideas are represented by the different targeting schemes discussed in this report.

We begin by developing SIMPLL, a simplex-type algorithm for globally optimally assigning shooters to targets. Among all the methods tested, this one is most demanding of situational awareness. SIMPLL rapidly solves the problem of (nearly) optimally assigning a fixed collection of shooters to a fixed collection of targets, given known kill probabilities for each potential assignment. The algorithm is represented by *SIMPLL.dll*, a Windows dynamic-linked library.

While there is only one way of making assignments optimally, there are a variety of suboptimal methods, some of which require much less information and coordination. Two of these (GREEDY and RANDOM) are compared with SIMPLL using *NetFire.xls*, an Excel workbook that implements all three methods simultaneously so that they can be compared on specific scenarios. SIMPLL is bound to win such a comparison, but the interesting question is whether the margin of victory is big enough to justify the advanced C4ISR system that SIMPLL requires. The answer is that the margin of victory depends strongly on circumstances. In cases where all targets are equal and all shooters have the same capability, there is little to be gained by global coordination. In more variable circumstances, the margin of victory can be large. Understanding the impact of circumstances is key to understanding the effects of internetted fire.

Introduction

Modern warfare is distinguished by the presence of increasing amounts of relevant information on the battlefield. The fog of war will never disappear, but it has thinned a bit due to continuing improvements in sensors, computers and communication networks. It is now realistic to talk about "internetting of fires" in the sense that targeting information used by a given shooter may come from sensors not under his direct control. But this internetting capability comes at a price, both fiscally and tactically. The sponsor of this research described the problem as follows:

> *The emerging Objective Force (OF) operational concepts embrace "internetting of fires" as an enabling concept. In further analysis, it may reveal itself to be vital to OF. Simply put, it is the ability to engage a particular target using any number (1 or more) of potential firers who are able to engage due to being on the network which provides targeting information. The target must first be sufficiently acquired to be engaged, and a decision must be made to actually engage it (target cleared to be engaged); this may depend upon humans, software or a combination of the two. The acquisition of a target by a sensor or sensors (e.g., sensor(s) on a UAV or main battle tank) may not necessarily result in an engagement by the platform(s) hosting those sensor(s), even if capable of engaging that target (e.g., armed UAV or main battle tank). Instead, some other firer on the network may engage the target. Through this "internetting of fires," the force is expected to achieve quicker times to engage and more efficient and/or effective allocation of available fires. An analytic tool is needed to conduct initial "screening" research of this concept, leading to a better understanding of the dynamics, critical variables, cause-and-effect relationships, and thereby permitting more sophisticated models to be built for further analysis, including their potential incorporation into force-on-force models. –Mike Baumann, director of TRAC, United States Army*

Our candidate for an initial screening tool is *NetFire.xls*, a workbook that compares multiple methods of coordinating fire. We postulate a group of targets, each of which has a value, and compare various targeting schemes on the basis of the Measure of Effectiveness (MOE) "fraction of total value killed, on the average". Clever schemes with a high investment in information systems can be expected to achieve high scores, but so can less advanced schemes that simply have more firepower. The incremental dollar should be spent in whatever manner increases the MOE the most.

The general concept in NetFire is that there is a certain length of time called a decision cycle within which no Battle Damage Assessment (BDA) is possible. Within that cycle, a fixed set of weapons ("shooters") is available to be fired at the enemy. NetFire reports results for several different methods of targeting, some methods requiring only local information and one (SIMPLL) requiring global information. The SIMPLL algorithm is a principal output of this research.

NetFire is an abstract combat model. The enemy does not move or shoot back, and, with one exception, time is not represented. "Combat" simply consists of one-off situations where dispersed shooters must simultaneously commit to targets. Nonetheless, NetFire can be used to explore the significance of certain kinds of information, specifically BDA information and coordinating information. The next section describes the analysis that underlies the calculations. The specifics of using the Excel workbook *NetFire.xls* are described afterwards.

Analysis of Fire Allocation Methods

Cases RANDOM, GREEDY, and OPT

Information may or may not be important to determining the battle outcome. Consider the following scenario. Suppose that there are *m* shooters and *n* targets, with each shooter having a single shot that is capable of killing its target with probability *Pk*. Temporarily make the following assumptions:

- *Pk* is the same number for all shooters and targets;
- the weapons are all "long range" in the sense that any shooter can physically attack any target;
- each target has the same value;
- each shooter possesses the required targeting information for each target; and
- there is no BDA between shots over the time period under consideration, so all shots might as well be committed in one salvo.

It does not follow from these assumptions that shooters can communicate with each other, nor does it follow that there is a commander capable of sensing the overall situation and giving appropriate orders. How much is this additional coordinating information worth?

The worst possible outcome from the viewpoint of the shooters would be if all *m* shots were directed against a single target, in which case the unfortunate target would be overkilled (unless *Pk* is very small), while *n*-1 targets would not be attacked at all. The optimal outcome in this case would be if all targets were shot at equally. Clearly, achieving the best outcome requires a commander with perfect information. However, the worst outcome *also* requires a commander with perfect information, albeit with a different motivation, so comparing the best with the worst will not be enlightening. Instead, we compare the best with an outcome where the information required to coordinate shooting at all targets equally is missing. If the shooters cannot communicate—either with each other or with the commander—then it makes sense to assume that the target chosen by one shooter is statistically independent of the targets chosen by other shooters. The resulting outcome will be intermediate, but represents an information extreme in the sense that no coordination whatever is required to achieve it. It is these two extreme situations, random versus optimal, that should be compared in discussing the value of coordinating information.

The optimal allocation will allocate the shots equally, so *m*/*n* shots will be assigned to each target. Each target survives only if all of these shots miss, so the average number of targets killed will be $E_{opt} = n(1-(1-Pk)^{m/n})$, or slightly less if *m*/*n* is not an integer. On the other hand, if each shot is directed at a random target, then each target survives a given shot with probability 1-*Pk*/*n*, independently of the other shots. Since there are *m* shots in total, the average number of targets killed will be $E_{rand} = n(1-(1-Pk/n)^m)$. Figure 1 shows both of these quantities as a function of *m* when *n* = 10 and *Pk* = 0.5.

.75

~.65

$E_{opt} = 10(1-.25) = 7.5 \quad N = 2$

$n = 10$

~~$E_{rand} = 10(1-.25)^2 = 5.6$~~

$E_{rand} = $ ~~$10(1-.5)$~~ $\frac{}{10}$

$= 10(1-(.95)^{20})$

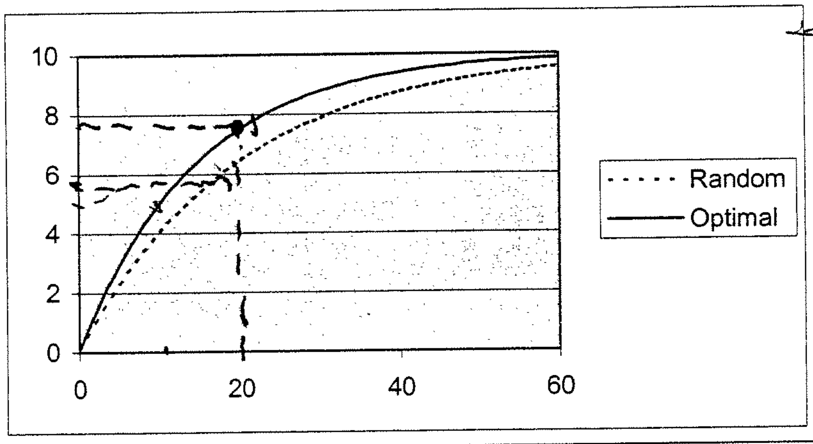


Figure 1: Average number killed versus total number of shots. Pk = .5

It can be seen from Figure 1 that there is very little difference between the optimal and random results when *m* is zero (in which case both quantities are of course 0) or very large (in which case both approach *n*). Even in the middle, where *mPk* and *n* are approximately equal, the functions are surprisingly close, given the extreme differences between the command and control systems being modeled. When all weapons are equally effective, all targets are equally valuable, and the kill probability is not too large, there turns out to be only a limited payoff for information that would permit the commander to make an optimal assignment for every weapon. The local tactic of assigning weapons uniformly at random is not that far from global optimality.

In mixed scenarios where target values differ and the kill probability depends on both weapon and target, the situation changes. It is no longer clear what firing strategy should be used in the extreme case of no coordination, since now there is a conflict between treating all targets equally and selecting the local target for which the kill probability is largest. To be more precise about this, as well as to account for the type of internetting information that makes firing even feasible, we introduce the following notation:

i indexes shooters, k indexes targets

V_k = value of target k

S_i = number of shots from shooter i

p_{ik} = probability that one shot of i kills k, given that i shoots at k

I_{ik} = 1 if it is feasible for i to shoot at k, else 0 {depends on weapon range, internetting, and possibly other command and control factors}

$P_{ik} = p_{ik} I_{ik}$ {only the product of p and I is important}

When there is no commander able to coordinate fires, each shooter could still choose a target at random. But a target truly chosen at random could very well have a zero kill probability, so instead we assume that the randomization is done only over those targets for which the kill probability is nonzero. To be precise, we define:

Case RANDOM: Assume that each shot from i is equally likely to be taken against any target k for which $P_{ik}>0$. Define the sets $T_i \equiv \{k|P_{ik}>0\}$ and $R_k \equiv \{i|P_{ik}>0\}$, and let #() count the number of elements in a set. Then $P_{ik}/\#(T_i)$ is the probability that each of i's independent shots kills target k, and therefore the probability that target k survives is $q_k = \prod_{i \in R_k} \left(1 - \frac{P_{ik}}{\#(T_i)}\right)^{s_i}$. The survival events are not independent among the targets, but nonetheless the expected remaining target value is $V^* = \sum_k V_k (1 - q_k)$.

Case RANDOM could be criticized for sometimes directing fire against targets where the kill probability is low, albeit positive. It is tempting instead to assume that shooter i directs all of his fire against the target for which the product of target value and kill probability is largest. This "GREEDY" procedure is also feasible without coordination, as long as target values can be sensed locally.

Case GREEDY: Assume instead that each of i's shots is taken against the target for which it is most effective, namely target $K(i) \equiv \text{argmax}_k \{V_k P_{ik}\}$, and let $R_k = \{i \mid K(i) = k\}$ be the set of shooters who attack target k. Then $q_k = \prod_{i \in R_k} (1 - P_{ik})^{s_i}$, and the expression for V^* is as before.

Case GREEDY may not do as well as Case RANDOM. Suppose that there were some single valuable target that is very vulnerable to fire from every shooter. Then all fire would be directed against that one target in Case GREEDY and all others would escape. Case GREEDY makes no provision for spreading fire over other targets, but rather relies on the inherent variability of kill probabilities and target values to accomplish this. If the variability is not there, GREEDY results will not be good.

The fact that neither RANDOM nor GREEDY is clearly the right thing to do when no coordination is possible simply bespeaks a difficult fire allocation problem. In this circumstance, one might suspect that a coordinating commander could make a significant difference, particularly when at least some of the kill probabilities are large. Our reasons for introducing the RANDOM and GREEDY procedures are not that we advocate them in any sense, but that they are simply two procedures that might characterize what actually happens in the absence of the coordination that internetting makes possible. Specifically, they might characterize target selection by "agents" whose behavior is based only on locally sensed conditions.

The best scenario for coordination information, then—the case where its value in terms of additional targets killed is largest—will be when:

- targets values are variable;
- kill probabilities are variable;
- at least some kill probabilities are large; and
- the scenario is not one of massive overkill or underkill.

It might seem that this is a rather narrow scenario, and in a way it is. Nonetheless, our intention is to concentrate on scenarios of this sort. Our contention is that this situation characterizes modern warfare. In particular, modern weapons may be guided weapons that are long range and have large kill probabilities, but which are in limited supply on account of being expensive. It is particularly important to employ such weapons carefully, reserving them for cases where less expensive weapons cannot do the job. Only coordinating information can enable this.

When coordinating information is available and timely commands to individual shooters are feasible, the fires of all the shooters can be directed in a manner that maximizes the average target value killed. The resulting optimization problem is a nonlinear assignment problem, since the objective will be best accomplished by assigning each shot to the target at which it is most effective.

Case OPT: Let the variable x_{ik} represent the number of shots made by shooter i at target k. The objective is to

maximize $\sum_k V_k(1-q_k)$, subject to the constraints

$$q_k = \prod_i (1-P_{ik})^{x_{ik}},$$

$$\sum_k x_{ik} \le S_i,$$

$$x_{ik} \ge 0$$

Here, and in all following optimization formulations, we use the convention that variable names begin with lower case letters, whereas constants begin with upper case letters. The formula for q_k reflects the usual independence assumption in the extended product of miss probabilities. The kill probability for target k is $1 - q_k$, so the objective function is "average value killed".

Case OPT is a nontrivial optimization problem, since the objective function is nonlinear and the variables are required to be integers. Murphey (2000) refers to it as the Static Weapon Target Assignment (SWTA) problem, and notes that it is *NP*-complete. We cannot solve it in reasonable amounts of time for large problems. However, an upper bound on the optimal objective function can be obtained by relaxing the integrality constraint, which produces a simpler, continuous optimization problem for which an efficient solution technique is known (Washburn[1995]). A lower bound can then be constructed by rounding the solution to integers in such a manner that shot constraints are still enforced. This process is carried out in *SIMPLL.dll*, a dynamic-linked library that is called by Visual Basic code within NetFire.xls. SIMPLL is capable of finding truly optimal integer solutions using a branch-and-bound technique (Washburn[1998]), but optimal solutions usually differ only slightly from the solution obtained by rounding the relaxed solution, and are prohibitively time consuming for large problems. Other methods of solving this problem are discussed by Ahuja, et all (2002).

Other Possible Views of the Fire Allocation Problem

The basic viewpoint taken above is that finite stocks of weapons must be allocated to targets within any given decision cycle. The weapons themselves are not "costly" in any sense other than being in limited supply. This is a common point of view, but by no means the only reasonable one. If weapons are costly, instead of being constrained, it may make sense to minimize the cost of "getting the job done" in some sense. This new point of view results in the consideration of optimization problems such as **NLP1**:

minimize $\sum_{i,k} C_i x_{ik}$, subject to the constraints

$$q_k = \prod_i (1-P_{ik})^{x_{ik}},$$

$$\sum_k V_k q_k \le V,$$

$$x_{ik} \ge 0$$

In NLP1, the weapon constraints S_i have been replaced by weapon costs C_i, and the object is to minimize the total cost of shooting, subject to the total surviving value not exceeding V. Costs may be in units of dollars, or may reflect the weight or volume of the weapons, which would be a consideration when moving weapons into a distant theater. Cost might also reflect tactical preferences.

NLP1 is a convex program, since the target value constraint amounts to requiring that a convex function not exceed V. NLP1 can therefore be solved by any of a number of commercially available nonlinear optimization packages. If the variables are required to be integers, a branch-and-bound algorithm could be based on an NLP1 relaxation, as in SIMPLL. However, there is no tailored algorithm for solving NLP1. We have investigated solutions to this problem using GAMS (Brooks, Kendrick, and Meeraus [1988]) to access general purpose solvers, and find that it solves much more slowly than Case OPT. For example, solving a 25 shooter, 75 target problem takes about a second for SIMPLL, but a minute for a general purpose solver that does not exploit the special structure. The performance spread increases as the problem size grows, highlighting the advantages of having a special-purpose tailored algorithm such as SIMPLL.

In principle, one could also solve **NLP2:**

minimize $\sum_{i,k} C_i x_{ik} + \sum_k V_k q_k$, subject to the constraints

$$q_k = \prod_i (1-P_{ik})^{x_{ik}},$$

$$x_{ik} \ge 0$$

NLP2 is the problem of minimizing total cost, including the "cost" of the surviving targets, subject to no constraints on either target value or weapon consumption. Weapon consumption is left to be whatever is natural for the target complex. NLP2 is a LaGrangian relaxation of Case OPT as well as of NLP1, so it should be easier to solve than either one. A special-purpose algorithm similar to SIMPLL could surely be developed for it. NLP2 suffers from the practical problem that both weapons and targets must be valued on the same scale, probably dollars. The tradeoffs that are implicit in Case OPT and NLP1 are explicit in NLP2.

One could also go in the opposite direction and add constraints to NLP1, rather than remove them. If every target is constrained to survive with only an input probability Q_k, one arrives at **NLP3**:

minimize $\sum_{i,k} C_i x_{ik}$, subject to the constraints

$$q_k = \prod_i (1-P_{ik})^{x_{ik}},$$

$$q_k \le Q_k$$

$$x_{ik} \ge 0$$

NLP3 is especially simple because q_k can be eliminated, after which the constraint set is:

$$\sum_i x_{ik} \ln(1-P_{ik}) \le \ln(Q_k)$$

$$x_{ik} \ge 0$$

Since the objective function and all constraints are linear, this is a Linear Program. The problem here is that one must give a survival probability for each target, with no tradeoffs possible between targets. The minimized objective function might be disappointingly large if Q_k were made small for some hard-to-kill target.

None of NLP1, NLP2, and NLP3 is implemented in *NetFire.xls*.

Case BDA

It was mentioned in the Introduction that NetFire does not model time, with one exception. Here we discuss that exception.

All of the previous calculations pertain to a "one-off" war that is one decision cycle in length, where all weapons are assigned without regard to the effects of other weapons. It is likely in such circumstances that some weapons will be assigned to targets that have already been killed by other weapons—a potentially significant waste if the weapons are expensive. In a multi-cycle war where BDA information about the status of targets is available, this waste could be prevented. To bound the value of such BDA information, we consider in this section a situation where there is no time pressure, and where every shot is followed by an instantaneous, perfect assessment of whether the target was killed or not. No real BDA system is this good and no war is so leisurely, but it is still of interest to see how much additional target value could be killed by weapons not previously wasted on dead targets.

The data P_{ik}, S_i, and V_k has the same meaning as before, but many policies are now feasible that were previously not. These new policies all involve shooting a single shot at some target, assessing its effects, and then, depending on the effects, shooting another single shot at some possibly different target, and so on. Each of these policies is a very complicated object, since all eventualities must be allowed for. Each might be expressed as a decision tree. For example, one policy might begin, "Fire weapon 3 at target 6. Then follow branch A if target 6 is killed, otherwise follow branch B". There are a great many possible such policies, but the number is finite. The problem of finding the one that maximizes the expected target value killed is therefore well defined, although the number of possible decision trees is large enough to make exhaustion an unattractive technique. See Glazebrook and Washburn (to appear 2004) for further information about such problems.

We know of no practical way of finding the optimal policy. Our goal here is merely to find an upper bound on the expected target value that such a policy might achieve. Toward that end, we define a collection of indicator random variables:

Y_{ik} =1 if target k is killed by weapon i (for each k, at most one of these can be 1),
Y_k = 1 if target k is killed, and
X_{ik} = 1 if weapon i is assigned to target k (for each i, at most one of these can be 1).

The firing policy will induce many correlations between these random variables, so independence assumptions among them are not appropriate, but the collection is still useful for formulating the problem of finding the optimal policy. We first note that $Y_k = \sum_i Y_{ik}$, and that the total value killed is $Z = \sum_k V_k Y_k$. The problem (call it **P1**) of finding the optimal policy can therefore be posed as maximizing z, the expected value of Z, subject to constraints

(1) $\sum_k X_{ik} \le S_i$ for all i with certainty,
(2) $Y_k = \sum_i Y_{ik}$ for all k with certainty,
(3) $Y_k \le 1$ for all k with certainty,
(4) other constraints.

The other constraints in P1 include the crucial relationship between X_{ik} and Y_{ik}, the essence of the firing policy. Whatever that relationship, it must certainly hold that $E(Y_{ik}) \le P_{ik}\, E(X_{ik})$. Now, using lower case letters for expected values of random variables, we can construct a relaxation of P1 that we name **P2**:

maximize $z = \sum_k V_k y_k$ subject to
(1) $\sum_k x_{ik} \le S_i$ for all i,
(2) $y_k \le \sum_i x_{ik} P_{ik}$ for all k, and
(3) $y_k \le 1$ for all k.

P2 is a relaxation of P1. This is because a relationship that is true with certainty will also be true on the average, because sums and expected values can always be interchanged, because (4) implies that $E(Y_{ik}) \le x_{ik} P_{ik}$, and because the rest of (4) has simply been omitted in P2. P2 is a simple Linear Program that provides an upper bound on what is achievable with any shoot-look-shoot policy. P2 has a direct interpretation where x_{ik} is the average number of weapons of type i used on target k, and y_k is the probability that target k is killed. (1) requires that the average number of weapons of type i used not exceed the number available, (2) requires that the effect of each shot not exceed P_{ik}, and (3) requires that target k be killed at most once.

This BDA upper bound is one of the computations made in *NetFire.xls*, using Excel's Solver to solve the Linear Program.

The AO Boundary

Hierarchic military organizations sometimes partition the target set into parts, with each unit in the hierarchy being responsible for one of the parts. Doing so simplifies command relationships and minimizes the need for communications. It may also be beneficial in avoiding some of the overkill errors that Case GREEDY is subject to, where too many shooters choose a soft or valuable target. However, the partition can also be viewed as the addition of constraints to Case OPT, which can only have the effect of reducing the objective function. Is the reduction significant? An easy way to find out is to partition the battlefield into two halves, with the shooters and sensors in one half being ineffective in the other. Each of the cases discussed above can be applied separately in each half, and the results added. By comparing the results with the cases where the battlefield is not partitioned, one can measure the significance of the partition, as in *NetFire.xls*.

NetFire.xls

JANUS Data

Since the importance of information and optimization is highly circumstance dependent, it is important that realistic data be used if correct conclusions are to result. Although *NetFire.xls* is itself unclassified, it does incorporate utilities for importing kill probability information from the JANUS combat model. As distributed, *NetFire.xls* is unclassified because it uses an unclassified dataset from JANUS.

Except for kill probability data, most user inputs are directly to the main sheet of *NetFire.xls*.

Line of sight computations and other terrain effects are not directly computed in NetFire.xls. They can be incorporated by indicator functions in the same manner that boundaries are handled: a miss probability defaults to one if there is no line-of-sight for a direct fire weapon.

The Main Sheet of NetFire.xls

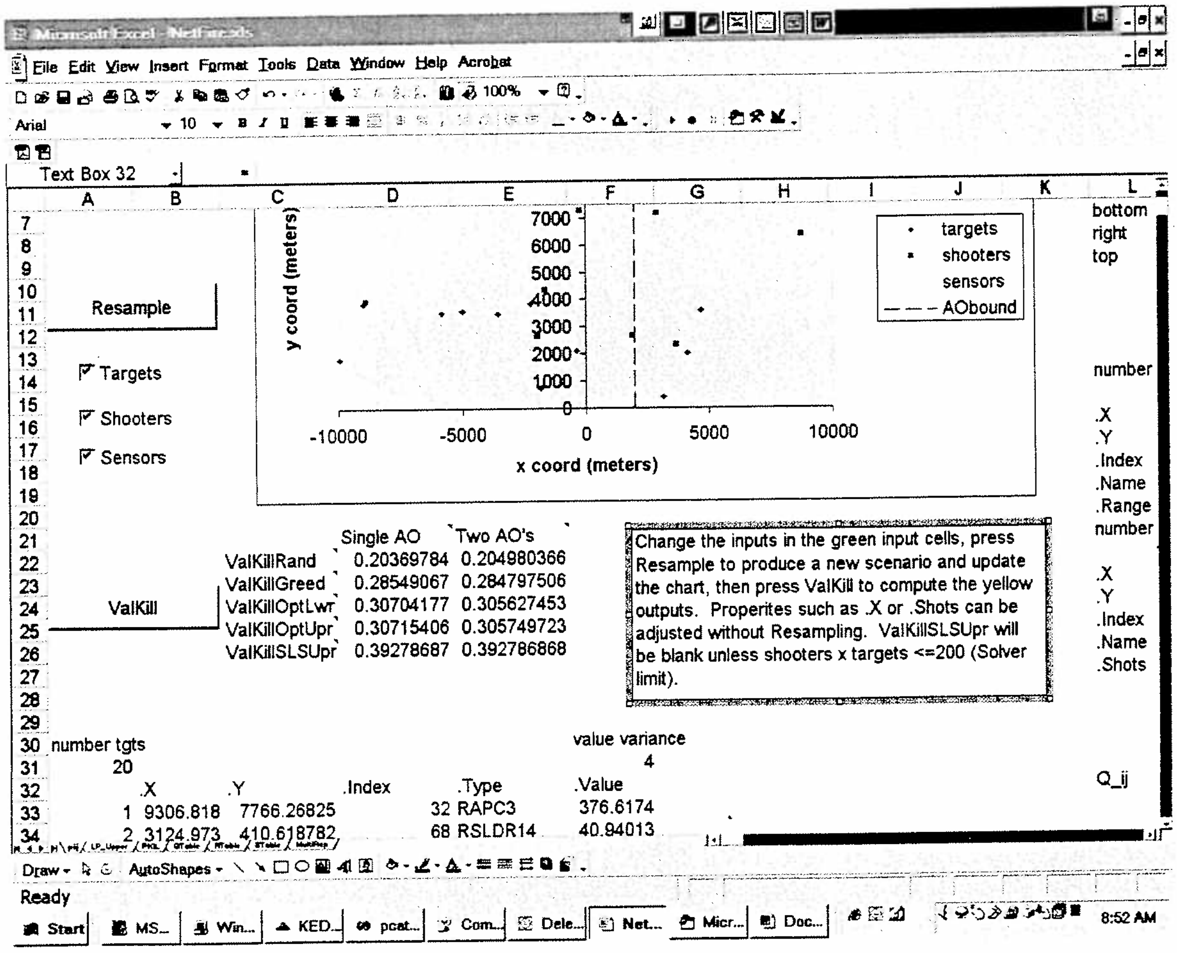
The main sheet of *NetFire.xls* is the sheet named "pij". On this sheet can be found a digital and graphical description of the current scenario, along with command buttons "Resample", "ValKill", and "MultiRep". These buttons have the following functions:

- The battle involves three types of objects: shooters, sensors and targets. Command "Resample" randomly selects and writes the object properties to the "pij" sheet, thereby defining a scenario. It also shows the object locations in a chart. The user's inputs here are the cells colored with a light green background: the number of each of the three types of object, the dimensions of the areas within which they are located, and, if desired, an AO boundary location. Resample also uses data from the sheets "QTable" (miss probabilities), "RTable" (max ranges), and "STable" (sensor data) to compute and write the miss probability matrices for both the AO case and the non-AO case. The user input cell "Value variance" increases the variability of the target values, with input 0 leaving them all at the default.
- "ValKill" computes the MOE "average fraction of target value killed" for four cases and two AO possibilities, writing values to the rows and columns of the output cells with a light yellow background. One method (OPT) has two rows because both a feasible solution and an upper bound need to be shown. ValKill uses the data in the ranges named "shots", "values", "missprob", and "missprobrl". Two tables of miss

probabilities are needed because results must be computed both with and without an AO. This data can be edited directly if desired, but will be overwritten when the Resample button is again pressed.

- "MultiRep" iterates the Resample/ValKill combination an input number of times, leaving only the last replication on sheet "pij". The 10 numbers computed by ValKill are written on the sheet "MultiRep" in rows, plus one additional row that averages the results in each column. The desired number of iterations is in the green cell right above the button.

Figure 2 shows a screen shot of the main sheet, including part of the map showing object locations and two of the command buttons. The check boxes are included because sometimes it is useful to resample only parts of the scenario, but normally all will be checked. The 10 numbers shown next to the ValKill button are the main outputs.



	Single AO	Two AO's
ValKillRand	0.20369784	0.204980366
ValKillGreed	0.28549067	0.284797506
ValKillOptLwr	0.30704177	0.305627453
ValKillOptUpr	0.30715406	0.305749723
ValKillSLSUpr	0.39278687	0.392786868

Change the inputs in the green input cells, press Resample to produce a new scenario and update the chart, then press ValKill to compute the yellow outputs. Properites such as .X or .Shots can be adjusted without Resampling. ValKillSLSUpr will be blank unless shooters x targets <=200 (Solver limit).

number tgts 20 — value variance 4

	.X	.Y	.Index	.Type	.Value
1	9306.818	7766.26825	32	RAPC3	376.6174
2	3124.973	410.618782	68	RSLDR14	40.94013

number .X .Y .Index .Name .Range number .X .Y .Index .Name .Shots Q_ij

Figure 2: Screen shot of the main sheet "pij" of workbook *NetFire.xls*.

Scenario Development in Resample

The Resample command button creates each scenario by first creating objects called shooters, sensors and targets with randomly selected properties. All such objects have locations (.X and .Y) that are uniformly sampled based on the limits given by the user on sheet "pij". These coordinates are shown on that sheet both digitally and graphically.

Each shooter also has a type that is selected with a sampling weight read from column 2 of sheet "QTable". These weights are all set to 1 when the sheet is created, but can be changed by the user if some shooters are desired to be more numerous than others. For example, make the weight of "PISTOL" 10 if Pistols should be 10 times as numerous as other shooters (only relative values of weights are significant, so arbitrary nonnegative values are permissible). Other editable items taken from "QTable" include:

- The number of shots available to each shooter (column 1).
- The value of each target (row 1).
- The sampling weight of each target (row 2).
- The miss probability for each (shooter, target) pair, conditional on a shot being feasible.

The maximum range for any given (shooter, target) pair is taken from sheet "RTable".

The Resample command also creates sensors, using data on sheet "STable". Each sensor has a name, a sampling weight, and a range within which it detects and identifies all targets. Given the user input number of sensors, the sensors are chosen randomly according to their weight, assigned the appropriate name and maximum range, and located uniformly at random within user-designated limits. As with shooter and target objects, the properties of each selected sensor are shown on sheet "pij" and may be adjusted by the user.

Finally, the Resample command creates two miss probability tables and shows them on sheet "pij". The first table is located in named range "missprob", and applies to the case where all targets are in the same area of operations (AO). The miss probability for any individual (shooter, target) pair is 1 if the shot is not feasible, or otherwise the value read from sheet "QTable". Feasibility requires that:

- The target must be within firing range of the shooter.
- The target must be known. A target is known if it is either
  - within range of any sensor, or
  - within VisRange of the shooter.

The VisRange parameter is currently hardwired at 3,000 meters in subroutine Sample(), which is the subroutine called by the Resample command to do all of the scenario generation work.

A separate table "missprobrl" is created for the AO case. All objects are partitioned by the AO boundary, and no shooter on one side of the boundary can even learn of a target on the other side, much less shoot at it. The AO miss probabilities are always larger than the non-AO miss probabilities, since shooters and sensors in one AO are not allowed to report on or shoot at targets in the other. It does not follow that results in the AO case will always be worse, although they usually are.

The Resample command allows the user to vary targets, shooters and /or sensors. This allows the analyst to obtain sufficient runs to estimate the effects of each with precision. For example, one can fix the sensor and shooter arrays, and investigate their performance against random target scenarios.

Computations Carried Out Within ValKill

All of the inputs required by the ValKill command are displayed on sheet "pij". In fact, the only data involved are in the named ranges "shots", "values", and "missprob" (non-AO) case, and "missprobrl" (AO case). All of this data can be edited directly before pressing the command button, but it will be overwritten the next time the Resample command button is pressed.

The function of the ValKill command is to compute the 10 numbers in the yellow shaded area, all of which are the MOE "average fraction of target value killed" in different circumstances. All of these computations are carried out within the ValKill() subroutine. The two local firing rules, RANDOM and GREEDY, are implemented directly within the subroutine. The OPT firing rule is implemented by calling subroutine SIMPL() in the external dynamic-linked library *SIMPLL.dll*. This subroutine returns an upper bound on what is achievable by any firing procedure faced with the same problem, so the upper bound is displayed, as well as the score achieved by the (nearly) optimal firing plan. The subroutine is declared at the beginning of module 1 of VBA code, and called from subroutine ValKill(). Finally, the BDA upper bound is calculated by calling subroutine LP_Upr(), which formulates and solves the appropriate Linear Program using the Excel Solver and sheet "LP_Upper". Solver will only handle problems with 200 or fewer variables. If the product of the number of shooters and the number of targets exceeds this limit, the upper bound is simply reported to be zero. This constraint could be eliminated by replacing the standard Solver with a better optimizer.

Each of the computations is repeated in the AO case, except that miss probabilities are taken from "missprobrl" instead of "missprob".

The SIMPL()subroutine in *SIMPLL.dll* can theoretically be called from any Windows application, as long as declarations are made carefully and the Compaq FORTRAN runtime library *DFORRT.dll* is also available.

NetFire was designed to generate random scenarios for analysis. It is also possible to use NetFire to analyze existing scenarios. To do so, first adjust the numbers of targets, shooters, and sensors to the correct values and press the Resample button. This will generate a random scenario with the correct dimensions. Then enter the correct target, shooter, and sensor data manually into the appropriate rows and columns. Save the workbook to protect all this work. Then uncheck the boxes for targets, shooters, and sensors under the Resample button on page "pij". Pressing the Resample button will then calculate the two required tables of miss probabilities. The ValKill button will then evaluate the different firing strategies.

In principle, NetFire could also be used as a fire planning tool. For a given target-shooter set, it will find the optimal allocation of fire to achieve the minimum expected surviving enemy force. This use for planning motivated the formulation of *SIMPLL.dll* as an independent subroutine.

General Tendencies and Conclusions

There are a few theoretical inequalities that must hold regardless of scenario. These will be stated in terms of $VK(i,j)$, the i^{th} row and the j^{th} column of the 10 outputs, $i = 1,..,5; j=1,2$. Table 1 shows a typical example.

1. $VK(i,j) \leq VK(4,j)$; $i =1,2,3$ and $j =1,2$. This is because an upper bound on the optimal

score must necessarily be greater than any feasible score, and all scores shown in the first three rows are feasible.

2. *VK(i,2)* ≤ *VK(4,1)*; *i* = 1,2,3. *VK(4,1)* is an upper bound on what can be achieved by any firing method that is required to make all shots within one cycle when there is no AO boundary, and therefore is also an upper bound when the AO constraint is added.
3. *VK*(4,*j*) ≤ VK(5,*j*); *j* = 1,2. One basically gets to the BDA bound by replacing the function *1-exp(-x)* in the OPT bound by the dominant function *min(1,x)*, so the BDA bound is necessarily at least as large as the OPT bound.

	Single AO	Two AOs
ValKillRand	0.2552852	0.245430418
ValKillGreed	0.26754648	0.245757911
ValKillOptLwr	0.31123148	0.276201159
ValKillOptUpr	0.31165106	0.276259909
ValKillSLSUpr	0.3365397	0.29697468

Table 1: Typical outputs from *NetFire.xls*.

All of the above inequalities should be true regardless of scenario. If any should ever fail to hold, then please notify one of the authors because a bug has been revealed.

Other than these inequalities, not much can be said in general about even the ordering of results, much less about absolute differences between them. It is not always true that GREEDY is better than RANDOM. To provide a counterexample, generate a scenario where all miss probabilities are equal. Every shooter will fire at the most valuable target in the GREEDY method, whereas the RANDOM method will spread fire equally over all, more or less. If the most valuable target is only slightly more valuable, RANDOM will get a better score. Also, in spite of inequality 2 above, it is not always bad to have two AOs. The GREEDY method, in particular, may do better when there is a constraint that prevents certain shooters from shooting at certain targets.

The incremental value of information is highly circumstance dependent, and there are so many dimensions to a NetFire scenario that one hesitates to make general conclusions beyond the above inequalities. Nonetheless, after generating and solving a large number of scenarios, any user will come to conclusions about tendencies. For the scenarios generated by the authors in the course of developing NetFire, the following tendencies have been noted:

- GREEDY generally outperforms RANDOM, but not by much.
- The OPT policy typically increases the fraction of target value killed by about 20%, relative to either GREEDY or RANDOM.
- The BDA upper bound is typically about 30% above the GREEDY/RANDOM level.
- A few percentage points are usually lost in going from one to two AOs.

The performance of BDA is surprisingly low, since BDA as used here represents a perfect assessment system where time is not constrained and the true status of every target is always reported correctly after every shot. As a means for promoting efficiency in firing, BDA appears to be not particularly effective, at least if the scenarios examined by the authors are typical. Of course, knowing the correct status of targets has a military payoff beyond weaponeering efficiency.

All of the above observations are based on 10 shooters with four shots each. For reference, the OPT policy with two shots each is approximately as effective as GREEDY/RANDOM with four shots. In this case, optimal use of information is approximately equivalent to doubling the firing assets available. If such results survive testing in more realistic scenarios, internetting of fires will truly be worth the cost.

The research original question from our sponsor asserted:

Through this "internetting of fires," the force is expected to achieve quicker times to engage and more efficient and/or effective allocation of available fires.

We find that the force does indeed achieve significantly more efficient and effective allocation of available fires when the force is internetted, but not under all circumstances. The question of whether internetting of fires provides *significant* improvements in *realistic* circumstances is worth pursuing further. This has significant implications for acquisition strategies as the Army allocates resources between C4ISR systems, weapons systems, and platforms.

References

Ahuja, R.; Jha, K.; Kumar, A.; Murphey, R.; (2002), "Exact Methods and Heuristics to Solve Weapon-Target Assignment (WTA) Problem", San Jose INFORMS meeting, 17-20 November.

Brooke, A., Kendrick, D., and Meeraus, A., (1988), *GAMS, a User's Guide*, Scientific Press.

Glazebrook, K., and Washburn, A. "Shoot-Look-Shoot: a Review and Extension", to appear in *Operations Research.*

Murphey, R., (2000), "Target-Based Weapon Target Assignment Problems", *Nonlinear Assignment Problems* (Pardalos and Pitsoulis,eds), Kluwer, chapter 3.

Washburn, A., "Finite Method for a Nonlinear Allocation Problem," (1995), *Journal of Optimization Theory and Applications*, Vol. 85, #3, pp. 705-726.

Washburn, A., "Branch and Bound Methods for a Search Problem," (1998), *Naval Research Logistics*, Vol. 45, #3, pp. 243-257.